**Purpose**

The purpose of this document is to propose an architecture and an API for communications between the scheduler and the information technology services (ITS) supporting the 412 Food Rescue app.

**Architecture**

The scheduler will run as an executable. It will have one command-line argument, -port, that specifies on what port it will offer its service (the default port is 8080). Once started, the scheduler will start an http service that supports communications between ITS and the scheduler in the form of POST messages and their corresponding responses.

**Communications Model**

The communications will be message-based, with each message sent from ITS having a corresponding response. All messages will have the standard POST and response headers with the content containing JSON strings. The top-level incoming messages to the scheduler support adding volunteer profiles, querying for transportation options for a request, deletion of volunteer profiles and requests, and commitments assigning requests to volunteers. The response for all of these message types includes status (whether or not the incoming message was properly processed), an error message if there is one, and a content representing the result of the message. Currently, the only message type with a content is the response to a request message, which includes a ranked list of transportation options.

The JSON objects in these messages are described below in the following order:

* Components classes – classes used among several message types
* Message and Response – the top-level classes for incoming messages and responses.
* Volunteer class – the class used to input a volunteer profile.
* Assignment – the class used to input a committed assignment of a volunteer to service a request.
* Request class – the class used to input a request, which generates a ranked list of assignment options
* Delete class – the class used to delete a volunteer profile or a request.

Finally, a control flow is described for what happens when the scheduler is started and when a new donation is assigned to a volunteer.

**Component Classes**

***Location***



Location represents a latitude/longitude point. The values are double precision floats.

***TimeWindow***



TimeWindow represents an interval of time. The values for start and end are 64-bit integers representing UNIX time (number of seconds that have elapsed since 00:00:00 Thursday, 1 January 1970).

**Message and Response**

***Message***



Message represents an incoming message to the scheduler. The value of id is an integer representing a unique identifier of the message. The value of messageType is an integer representing the type of the message. The value of content is a subclass of MessageResponseType and must correspond to the type. The subclasses of MessageType are described below.

***MessageResponse***



MessageResponse represents a response to an incoming message to the scheduler. The values of type and ID must match those in the incoming message. The value of status is a string that is either success, failure, or error. The value of content is either null or a subclass of MessageResponseType and is dependent on the type and the processing of the incoming message. If an exception occurs in processing the incoming message regardless of its type, then the content is error and errorString is set to an explanatory string. The subclasses of MessageResponseType are described below as part of the definitions of the incoming messages.

**Volunteer**

***Availability***



Availability represents a range of time and a corresponding location of when and where a volunteer is available to perform a pickup.

***Volunteer : MessageType***



Volunteer represents the profile of a volunteer. The ID is a unique identifier for the volunteer. The availabilities are an array of time-location pairs specifying when a volunteer is available to perform a pickup and where the volunteer is expected to be at that time. timeSinceLastRequest is the number of seconds since the last time the volunteer was offered an opportunity to transport a donation. autoNegotiate? is either true or false, depending on whether or not the volunteer is willing to allow the app to auto-negotiate.

**Assignment**

***Assignment : MessageType***



Assignment represents a time range when a volunteer can service a request. requestID is the unique identifier for the request. volunteerID is the unique identifier for the volunteer. pickupTimeWindow is the range in time when the pickup is expected to be made.

**Request**

***LocationUpdate***



LocationUpdate represents the current position of a manual (non-auto-negotiate) volunteer. The volunteerID is the unique identifier of the volunteer, and location is the latitude-longitude coordinate of the volunteer.

***Request : MessageType***



Request represents an order for a volunteer to transport a donation. ID is the unique identifier for the request. pickUpLocation and dropOffLocation are the latitude/longitude coordinates of the locations for picking up and dropping off the donation. pickUpTimeWindow and dropOffTimeWindow are the time ranges in which the donation can be picked up and dropped off. requiredCapacity is the number of capacity units needed to transport the donation. maximumTransportationTime is the longest time the donation can remain in the vehicle. manualNegotiatorsLocations is an array of LocationUpdate that provide all the current locations of the manual volunteers that are willing to service this request.

***RequestResponse : MessageResponseType***



RequestResponse represents the ranked list of options that are returned as a result of submitting a Request message to the scheduler. assignmentOptions is an array of assignments specifying pairs of volunteers and recommended time windows for making the pickup.

**Delete**

***Delete : MessageType***



Delete represent the message for deleting either a volunteer or a request from the scheduler’s memory. objectType is the type of object being deleted, i.e., the type for either a volunteer or a request. ID is the unique identifier for either the volunteer or the request.

**Control Flow**

**Initialization**

1. The scheduler is started and the volunteer profiles are loaded via a series of Volunteer messages.
2. Any existing assignments for the volunteers is added via a series of Assignment messages.

**New Donation**

1. A donation is assigned a non-profit destination and submitted to ITS to assign a volunteer to perform the transportation.
2. ITS solicits manual bids for a set period of time and collect them.
3. ITS creates a request and sends it to the scheduler. The request’s temporal, spatial, and capacity constraints are based on the donation and the pickup and dropoff locations. The manualNegotiatorLocations attribute is assigned LocationUpdates corresponding to the collected bids.
4. The scheduler returns the ranked list of options back to ITS via a RequestResponse message.
5. ITS iterates down the ranked list of options until it can secure a commitment from a volunteer. For an option for a manual volunteer, a commitment is automatically assumed. For an auto-negotiated volunteer, a commitment request is sent to the volunteer to accept or deny within a response period of time. If no response is received within the time window, the request is assumed to be denied.
6. Once a commitment is secured, the scheduler is informed of it via an Assignment message.